# Expérience

## Introduction

Dans cette troisième partie du travail de bachelor, je vais mener une expérience avec l’application que j’ai créée dans la deuxième partie. Cela va me permettre de tester le bon fonctionnement de cette application et aussi d’expérimenter les réseaux de neurones.

## Twitter

Nous avons à notre disposition un ensemble de tweet qui sont de nature positif ou négatif concernant des films. Notre ensemble est composé de 10662 tweets, les 5331 premiers tweets sont de polarité négative, la deuxième moitié est de polarité positive.

Dans le cas de notre expérience nous allons utiliser 90% de données pour l’entrainement et 10% pour la validation.

Un mélange des données sera effectué avant leur distribution entre l’entrainement et validation.

## Ensemble de données

Plusieurs ensembles de données ont été utilisée dans le cadre de cette expérience, nous allons les décrire dans ce chapitre.

### Twitter1

Dans twitter1 nous utilisions un vecteur one-hot, un vecteur one-hot est un vecteur unidimensionnel qui va permettre de savoir si un mot est contenu dans une phrase.

Pour générer un vecteur one-hot il faut premièrement créer un dictionnaire. Le dictionnaire va contenir tous les mots présents dans notre ensemble de données. La taille du dictionnaire va aussi représenter la taille de nos one-hot vecteur.

Pour chaque donnée nous créons un one-hot vecteur. Pour chaque mot contenu dans la donnée nous allons mettre la value à 1 dans le one-hot vecteur en fonction de la position de ce mot dans le dictionnaire.

Voici un petit exemple pour illustrer le concept :

Dictionnaire :

|  |
| --- |
| All |
| Was |
| A |
| Do |
| Making |
| Because |
| We |
| This |
| What |
| Must |
| Can |
| Good |
| Triumph |

Par conséquent la phrase « This was a triumph » va nous donner le vecteur suivant :

|  |  |
| --- | --- |
| All | 0 |
| Was | 1 |
| A | 1 |
| Do | 0 |
| Making | 0 |
| Because | 0 |
| We | 0 |
| This | 1 |
| What | 0 |
| Must | 0 |
| Can | 0 |
| Good | 0 |
| Triumph | 1 |

Avec cette méthode on espère que le réseau de neurone va déterminer si la présence de certain mot dans la phrase peut influencer le côté positif ou négatif.

Du coup l’ensemble de données twitter1 sera composé de vecteur one-hot. Dans notre cas seulement les noms seront retenus. Ce qui correspond à un dictionnaire de 9210 mots différents pour 10662 données. Ce qui va nous donner 98197020 valeurs.

### Twitter2

Twitter2 reprends les mêmes principes que twitter1 avec le one-hot vecteur sauf que cette dans cet ensemble de données nous prenons en comptes les verbes en plus des noms.

Ce qui nous donne un dictionnaire avec 11000 mots donc un total de 117282000 valeurs pour les 10662 données.

### Twitter3

Dans l’ensemble de donnée twitter3 on va essayer une approche différente que le one-hot vecteur.

L’idée est de se passer de dictionnaire, à la place on va convertir les lettres en nombre grâce à la correspondance entre une lettre et le code ascii.

Par conséquent on va se retrouver avec des vecteurs de nombre fortement réduit par rapport à un vecteur one-hot.

Dans notre cas la taille du vecteur sera en fonction du tweet qui contient le nombre le plus élevé de caractères.

Par exemple, la phrase « i like it » va générer le vecteur suivant :

|  |  |
| --- | --- |
| i | 105 |
|  | 20 |
| l | 108 |
| i | 105 |
| k | 107 |
| e | 101 |
|  | 20 |
| i | 105 |
| t | 116 |

(105, 20, 108, 105, 107, 101, 20, 105, 116), Il faut encore normaliser le vecteur pour que les valeurs se trouvent entre 0 et 1. Par conséquent on utilise la valeur maximal et minimal pour redimensionner les valeurs. La formule est la suivante : (x-20) / (116-20). Ce qui nous donne le vecteur (0.885, 0, 0.916, 0.885, 0.906, 0.843, 0, 0.885, 1).

Dans l’ensemble de données de 10662 tweets, le nombre maximum de caractères est de 216. Ce qui nous donne 2302992 valeurs.

### Twitter4

Twitter4 reprend l’idée de twitter3 excepté que à la place de convertir des lettres en chiffre on va directement convertir des mots en chiffre.

Cette fois nous n’allons pas utiliser la table ascii pour faire la conversion mais nous allons utiliser la fonction native de hachage de python.

La fonction permet de retourner un nombre entier relatif.

Voici un exemple de fonctionnement avec la phrase « the cake is a lie » :

|  |  |
| --- | --- |
| The | 332310010 |
| Cake | 1867153272 |
| Is | -1500461686 |
| A | -468864544 |
| Lie | 916042657 |

On obtient le vecteur (332310010, 1867153272, -1500461686, -468864544, 916042657). Après l’utilisation de la formule de normalisation on se retrouve avec le vecteur (0.544, 1, 0, 0.306, 0.717).

Comme dans twitter1 nous utilisons seulement les noms pour les mots les autres ne seront pas pris en compte.

Du coup la phrase qui contient le plus de mot dans notre ensemble de tweets possède 52 mots. Ce qui nous donne 554424 valeurs.

### Twitter5

Dans twitter5 on essaye une nouvelle méthode de représentation de données.

Jusqu’à maintenant nous avons représenté nos données avec seulement une dimension. Dans cet ensemble de données nous allons tester de représenter nos données avec deux dimensions.

Pour ce faire nous allons reprendre la méthode de twitter4 pour la représentation des mots en chiffre. Pour rappels nous avons 62 mots au maximum par tweet, nous allons donc utiliser une matrice 62x62.

La phrase sera présente une fois dans la première ligne et une fois dans la première colonne.

Par exemple pour la phrase « Be happy and smile » :

|  |  |  |  |
| --- | --- | --- | --- |
| Smile |  |  |  |
| And |  |  |  |
| Happy |  |  |  |
| Be | Happy | And | Smile |

Dans le cas des intersections entre deux mots nous allons additionner leur valeur en chiffre pour obtenir un nouveau nombre qui représentera l’addition des deux mots.

Si jamais l’intersection se passe dans la diagonale nous n’additionnons pas les deux mots. On garde seulement la valeur du mot d’origine.

|  |  |  |  |
| --- | --- | --- | --- |
| Smile | Smile + Happy | Smile + And | Smile |
| And | And + Happy | And | And + Smile |
| Happy | Happy | Happy + And | Happy + Smile |
| Be | Happy | And | Smile |

Maintenant on applique la conversation grâce à la fonction de hachage ce qui nous donne :

|  |  |  |  |
| --- | --- | --- | --- |
| -296382407 | 301078392 | -1885307975 | -296382407 |
| -1588925568 | -991464769 | -1588925568 | -1885307975 |
| 597460799 | 597460799 | -991464769 | 301078392 |
| 603887297 | 597460799 | -1588925568 | -296382407 |

Après normalisation la matrice nous donne :

|  |  |  |  |
| --- | --- | --- | --- |
| 0.638 | 0.878 | 0 | 0.638 |
| 0.119 | 0.359 | 0.119 | 0 |
| 0.997 | 0.997 | 0.359 | 0.878 |
| 1 | 0.997 | 0.119 | 0.638 |

A la fin on se retrouve donc avec 40984728 valeurs. On espère que ce format de donnée va permettre à la convolution de faire des relations entre des parités.

### Twitter6

Jusqu’à maintenant nous avons essayé de représenter les lettres ou les mots en chiffre via des fonctions de hachage ou des conversions vers la table ascii. Le défaut de ses méthodes c’est que des mots au sens proche seront représenter par des valeurs totalement différentes et on ne pourra pas obtenir de corrélation entre eux.

L’idée dans l’ensemble de données Twitter6 est de présenter chaque mot par un vecteur de réels et les mots apparaissant dans des contextes similaires auront des vecteurs plus proches que d’autres apparaissant dans des contextes différents.

On va s’appuyer sur des réseaux de neurones pour construire ces vecteurs. Un des architectures possibles est word2vec.

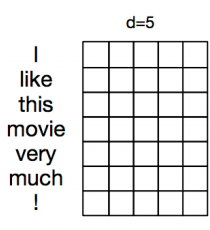


On peut voir dans l’image que le réseau arrive à ajuster des poids en fonction de catégorie.

La deuxième étape de cet ensemble de donnés est de représenter les données en deux dimensions.

Dans le cas d’un phrase les mots vont représenter la dimension horizontale et la phrase va représenter la dimension verticale.

Par exemple pour la phrase « I like this movie very much ! » en considérant qu’on représente un mot avec 5 valeurs par vecteurs.



On espère que cette méthode va permettre au réseau de neurone de faire des relations sémantique et syntaxique.

Dans le cas de twitter6 on utilise 61 mots par phrase (On ignore les mots qui ne sont pas répété plus de 5 fois dans la totalité des données), les mots sont représentés par des vecteurs de taille 100 et word2vec à entrainer son réseau de neurones pendant 5 itérations.

### Twitter7

Twitter7 reprends twitter6 excepté qu’on utilise 50 itération à la place de 5 pour l’entrainement de l’architecture word2vec.

### Twitter8

Twitter8 reprends twitter6 excepté qu’on utilise 200 itération à la place de 5 pour l’entrainement de l’architecture word2vec.

### Twitter9

Twitter9 reprends twitter6 excepté qu’on utilise 500 itération à la place de 5 pour l’entrainement de l’architecture word2vec.

### Twitter10

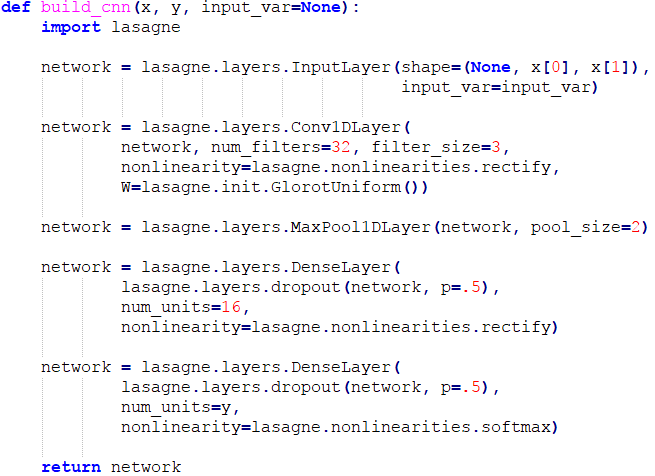
Twitter10 reprends twitter6 excepté qu’on utilise 200 itération à la place de 5 pour l’entrainement de l’architecture word2vec et on représente les mots par des vecteurs de 10 à la place de vecteur de 100.

### Twitter11

Twitter11 reprends twitter6 excepté qu’on utilise 500 itération à la place de 5 pour l’entrainement de l’architecture word2vec et on représente les mots par des vecteurs de 300 à la place de vecteur de 100.

## Architectures

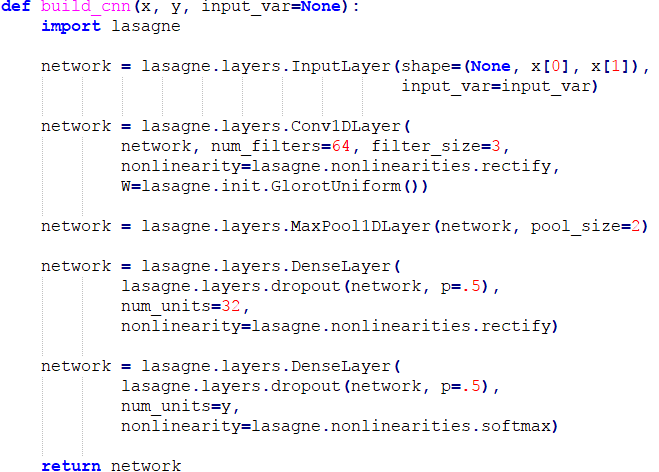
### Twitter1



Twitter1 est composé d’une couche de convolution avec 32 filtres de taille 3, suivis d’un pool qui va réduire par 2 les entrées. Pour terminer une couche entièrement connectée de 16 unités.

Hypothèse : On va utiliser la couche de convolution pour essayer de repérer les triplets de mot. On suppose que la couche de pool va réduire la distance entre les mots et permettre à la couche entièrement connecté de trouver la différence entre un tweet positif ou négatif.

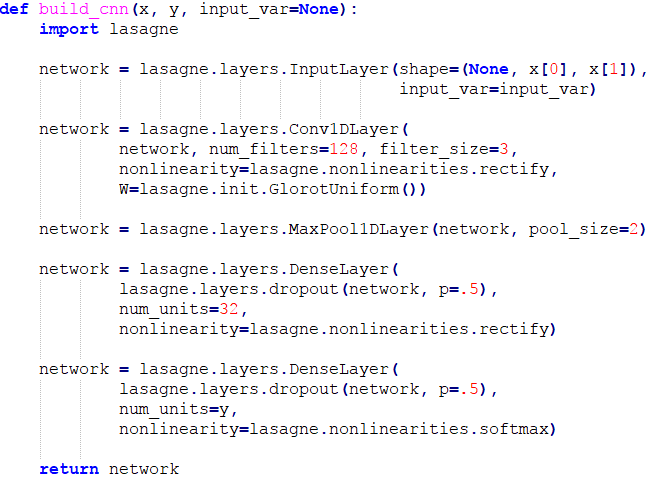
### Twitter2



On double la taille des filtres et de la couche entièrement connectée par rapport à twitter1.

Hypothèse : On augmente la taille des filtres et de la couche entièrement connectée pour être plus précis dans la reconnaissance des triplets positifs ou négatifs.

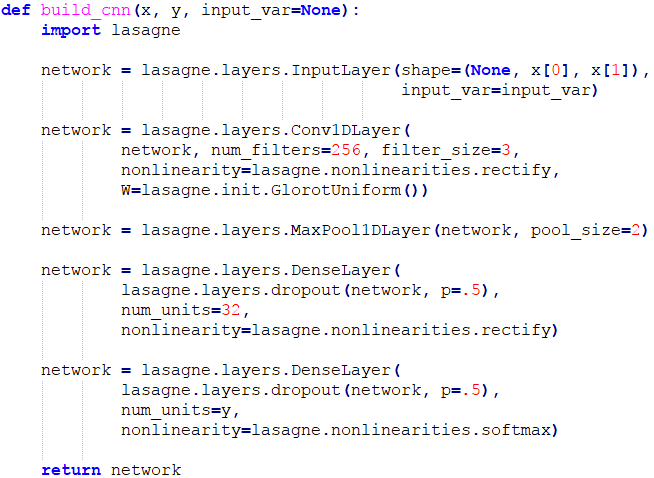
### Twitter3



On double la taille des filtres par rapport à twitter2.

Hypothèse : On suppose que l’augmentation du nombre de filtre va permettre aussi d’augmenter les performances.

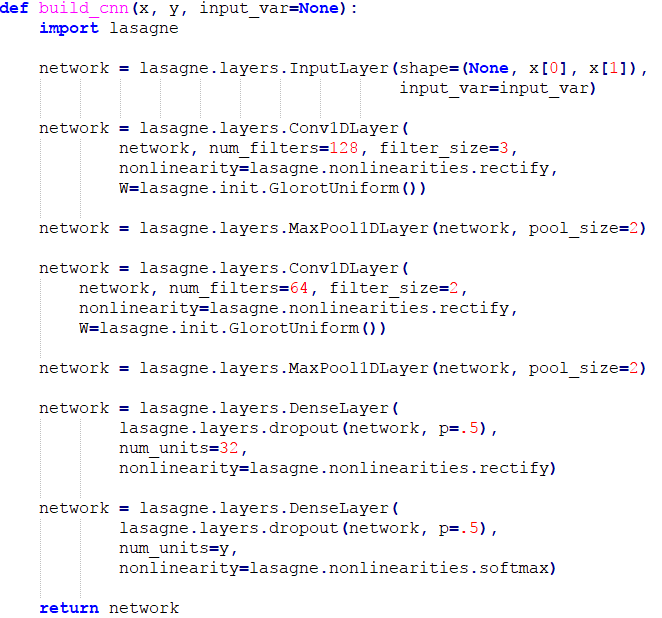
### Twitter4



On double la taille des filtres par rapport à twitter3.

Hypothèse : On suppose que l’augmentation du nombre de filtre va permettre aussi d’augmenter les performances.

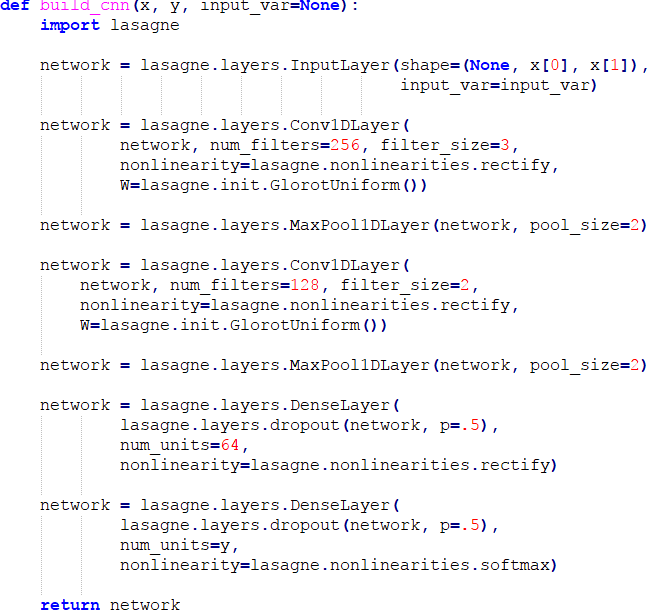
### Twitter5



On rajoute une couche de convolution avec 64 filtres de taille 2 et une couche de pool par rapport à twitter3.

Hypothèse : On suppose que l’ajout de la convolution de taille 2 va permettre de trouver des tuples de mot et ainsi augmenter la précision.

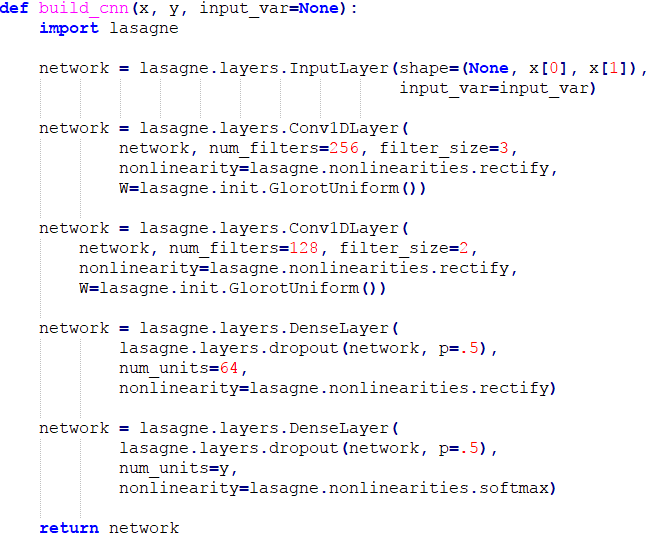
### Twitter6



On double la taille des filtres et de la couche entièrement connectée par rapport à twitter5.

Hypothèse : On suppose que l’augmentation du nombre de filtre et la taille de la couche entièrement connectée va permettre aussi d’augmenter les performances.

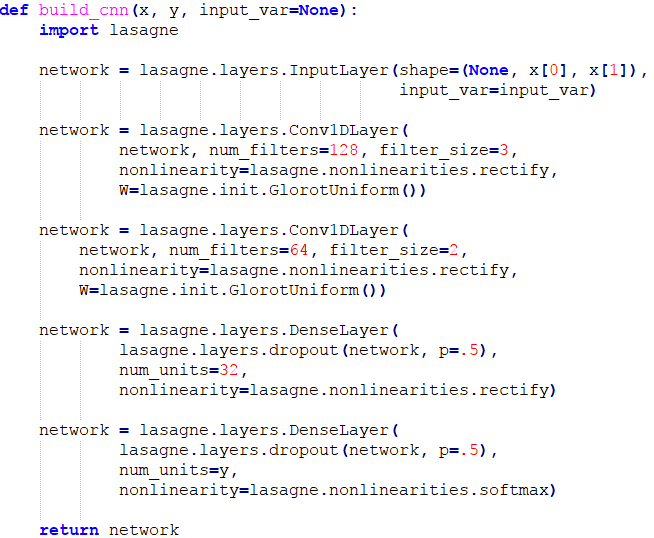
### Twitter7



On supprime les deux couches de pool par rapport à twitter6.

Hypothèse : Une suppose que la réduction des données par le pool supprime trop d’information et par conséquent réduit les performances de l’architecture.

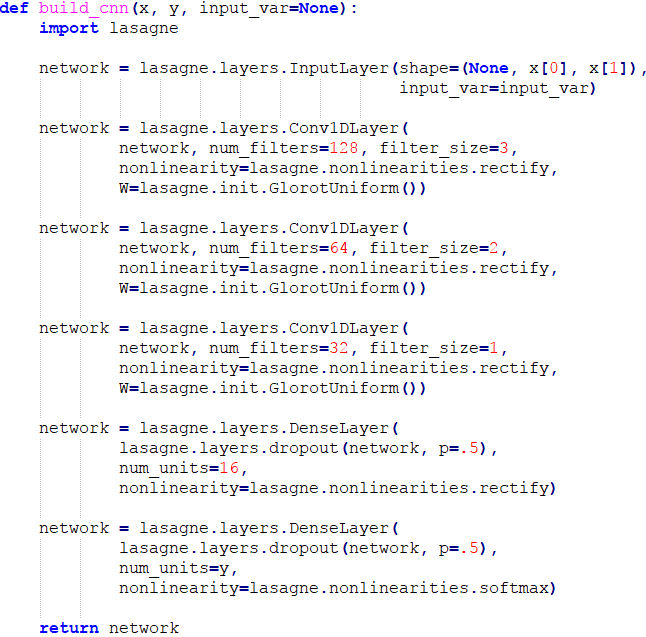
### Twitter8



On divise par 2 les filtres et la couche entièrement connecté par rapport à twitter7.

Hypothèse : On réduit le nombre de filtre pour voir si un grand nombre de filtre empêche le réseau de neurone de tirer des conclusions.

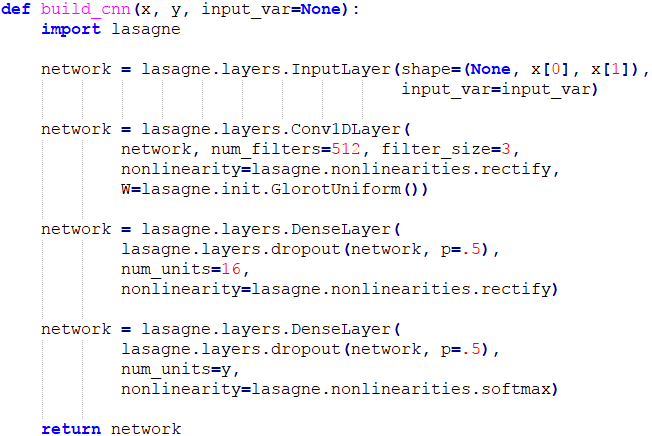
### Twitter9



On rajoute une couche de convolution avec 32 filtres de taille 1 et on divise par 2 la taille de la couche entièrement connectée par rapport à twitter8.

Hypothèse : On suppose que l’ajout de la couche de convolution de taille 1 va permettre d’augmenter la précision en reconnaissant la présence de certain mot.

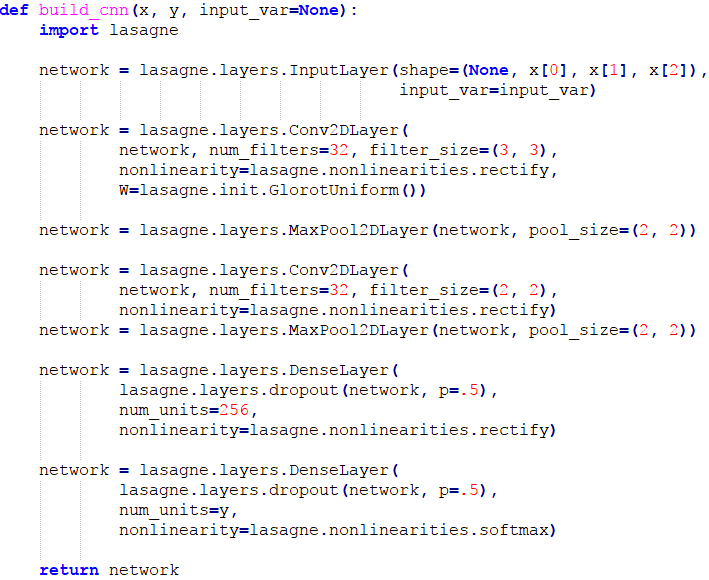
### Twitter10



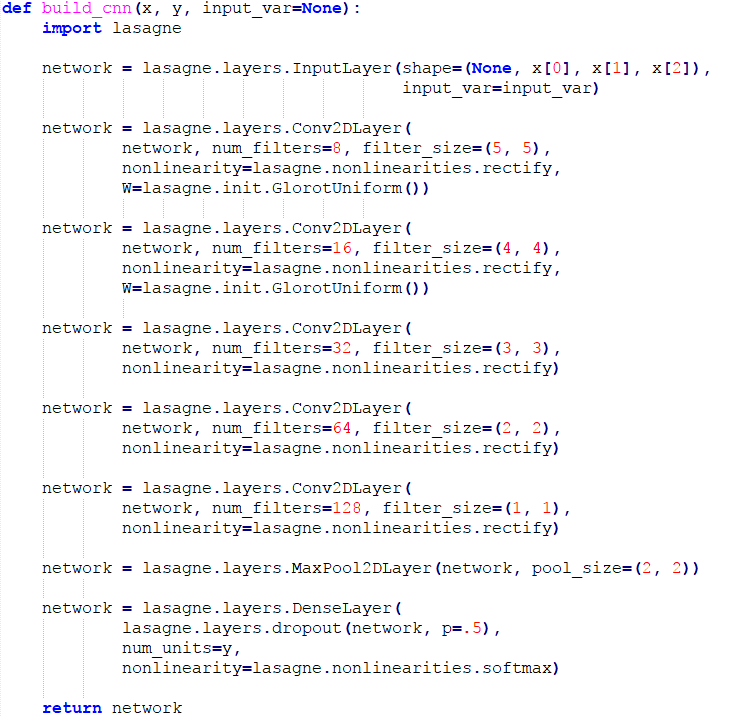
On enlève la couche de pool et on quadruple la taille des filtres par rapport à twitter1.

Hypothèse : On suppose qu’une couche avec seulement les triplets avec un très grand nombre de filtre va permettre d’augmenter les performances.

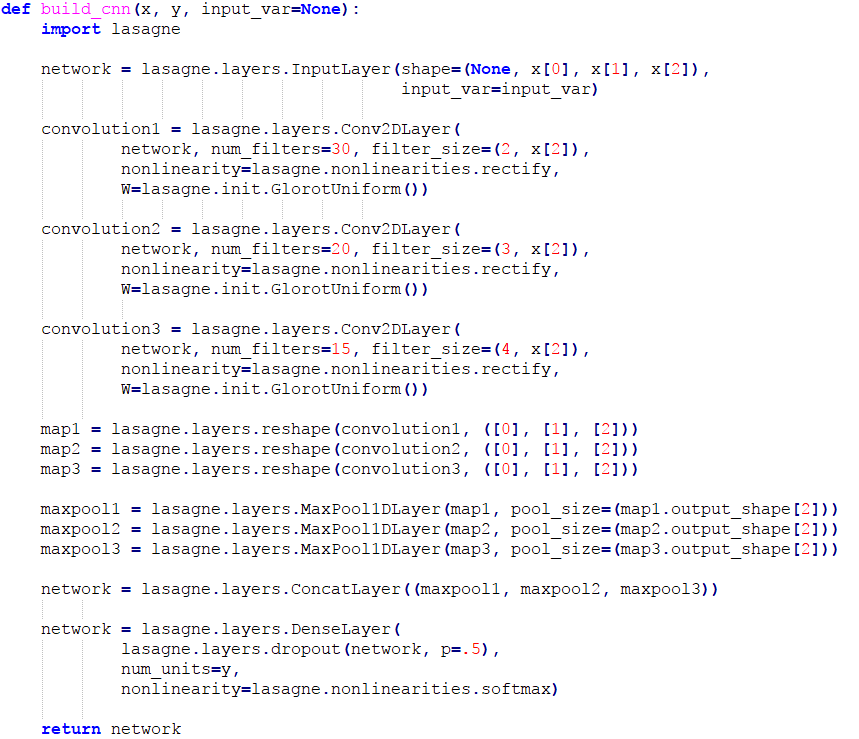
### Twitter11



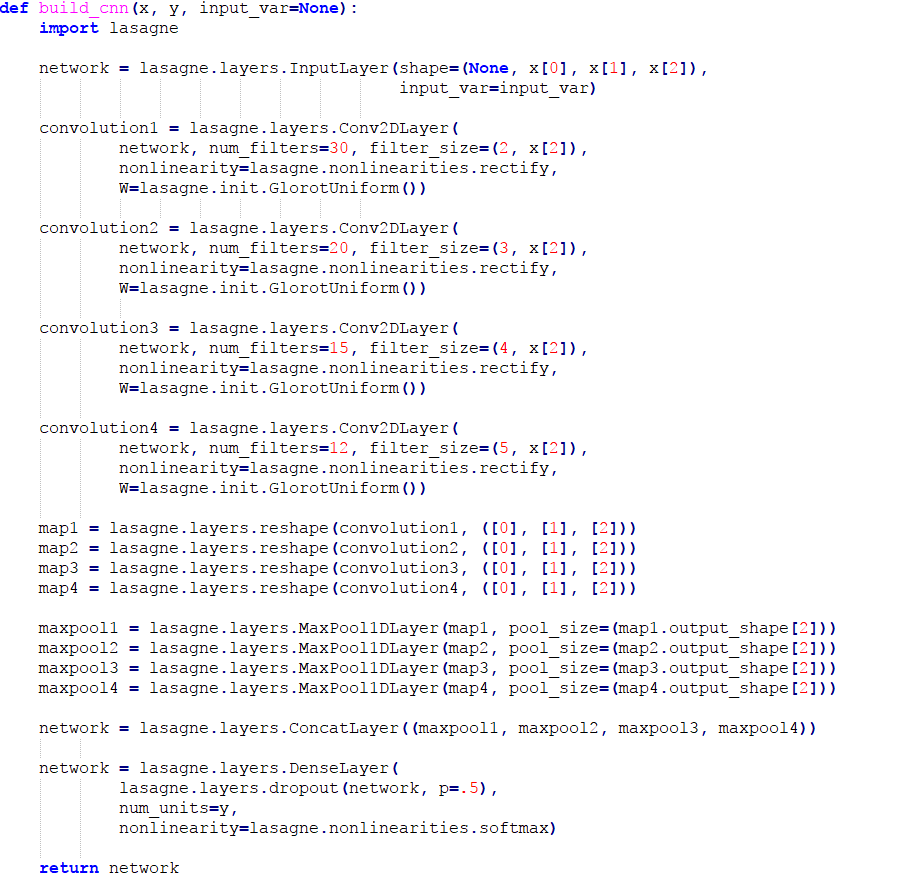
### Twitter12



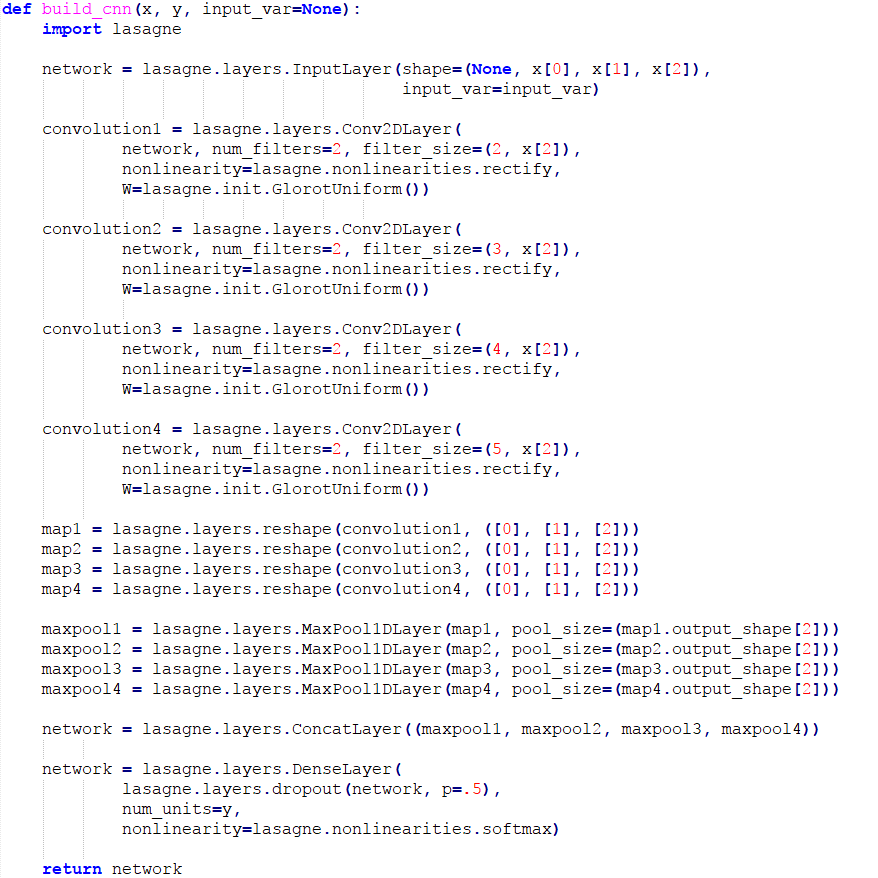
### Twitter13



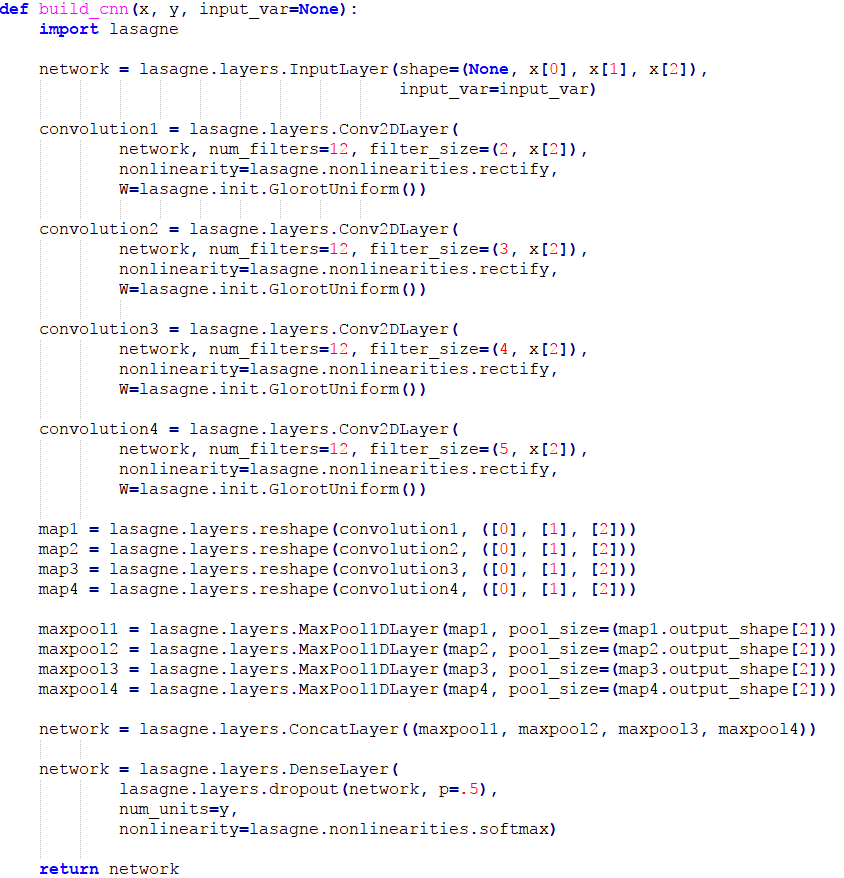
### Twitter14



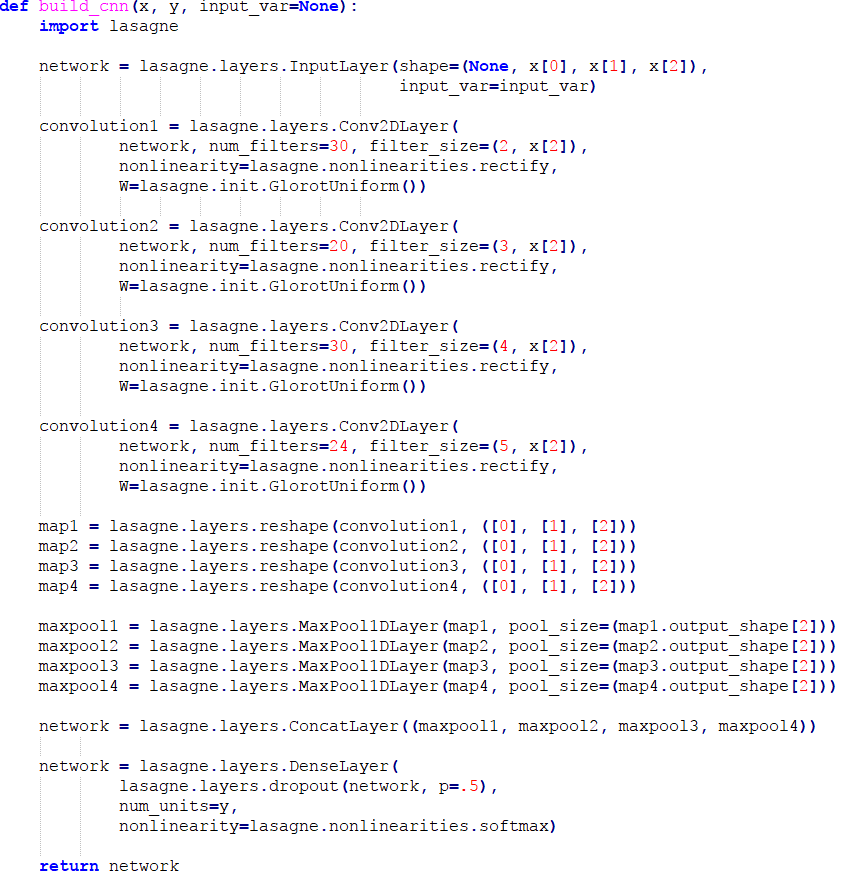
### Twitter15



### Twitter16



### Twitter17



## Résultats

Le tableau suivant contient les résultats pour les ensembles de données et architectures contenant une seule dimension.

Les colonnes représentent les ensembles de données et les lignes représentent les architectures.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | T1 | T2 | T3 | T4 | AVG |
| T1 | 71.30 | 71.40 | 53.40 | 52.70 | **62.20** |
| T2 | 70.40 | 71.80 | 53.10 | 51.90 | **61.80** |
| T3 | 70.90 | 71.20 | 52.70 | 51.40 | **61.55** |
| T4 | 69.70 | 71.90 | 52.20 | 51.60 | **61.35** |
| T5 | 68.90 | 67.50 | 53.30 | 51.50 | **60.30** |
| T6 | 68.70 | 68.40 | 53.10 | 51.80 | **60.50** |
| T7 | 69.90 | 72.60 | 52.70 | 53.40 | **62.15** |
| T8 | 70.80 | 72.10 | 52.70 | 52.30 | **61.98** |
| T9 | 71.40 | 70.60 | 53.30 | 50.80 | **61.53** |
| T10 | 71.00 | 72.20 | 52.70 | 53.40 | **62.33** |
| AVG | **70.30** | **70.97** | **52.92** | **52.08** |  |

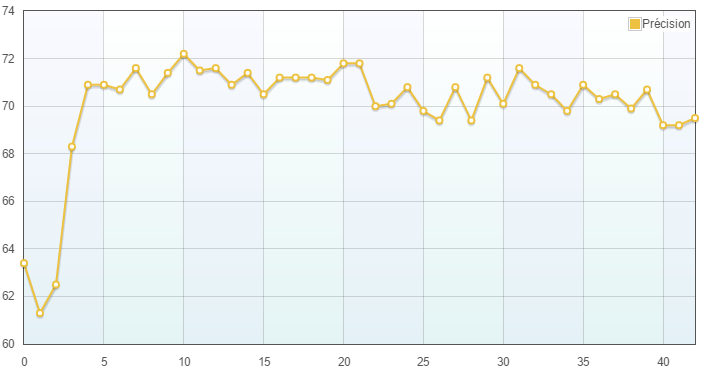
Les meilleurs résultats en fonction des ensembles de données.

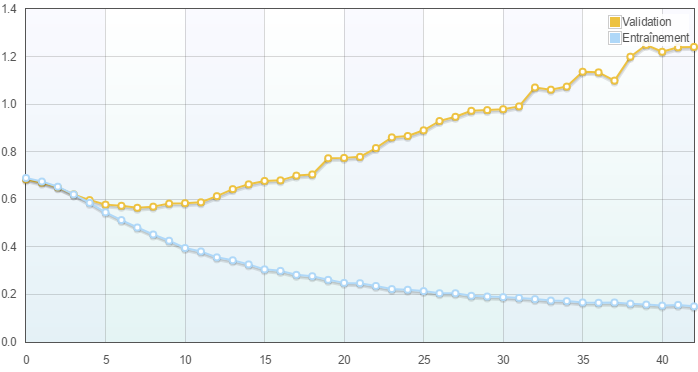
|  |  |
| --- | --- |
| T2 | 70.97 |
| T1 | 70.30 |
| T3 | 52.92 |
| T4 | 52.08 |

Les meilleurs résultats en fonction des architectures.

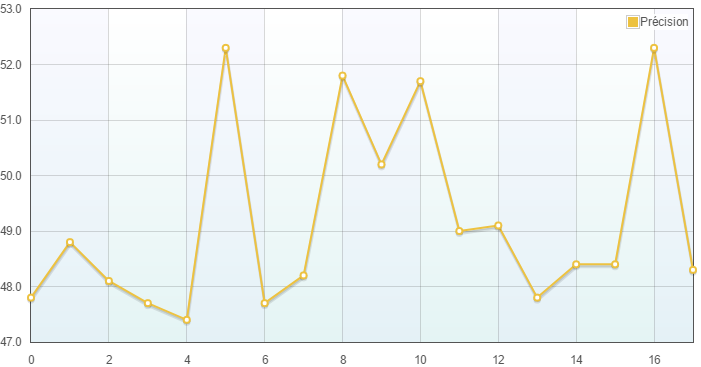
|  |  |
| --- | --- |
| T10 | 62.33 |
| T1 | 62.20 |
| T7 | 62.15 |
| T8 | 61.98 |
| T2 | 61.80 |
| T3 | 61.55 |
| T9 | 61.53 |
| T4 | 61.35 |
| T6 | 60.50 |
| T5 | 60.30 |

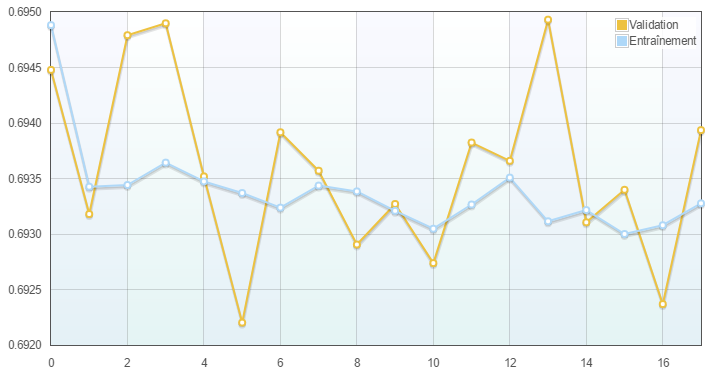
Représentation graphique de la meilleure architecture avec le meilleur ensemble de données (twitter10 avec twitter2).





Représentation graphique de l’architecture twitter5 avec l’ensemble de données twitter2.





Le tableau suivant contient les résultats pour les ensembles de données et architectures contenant deux dimensions.

Les colonnes représentent les ensembles de données et les lignes représentent les architectures.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | T5 | T6 | T7 | T8 | T9 | T10 | T11 | AVG |
| T11 | 52.30 | 57.50 | 68.40 | 72.70 | 72.20 | 66.40 | 71.10 | **65.89** |
| T12 | 52.20 | 57.40 | 66.70 | 72.90 | 68.40 | 67.00 | 70.50 | **65.01** |
| T13 | 51.70 | 57.50 | 74.90 | 77.80 | 76.60 | 70.90 | 80.00 | **69.91** |
| T14 | 52.90 | 58.50 | 74.70 | 77.00 | 77.40 | 70.20 | 79.30 | **70.00** |
| T15 | 52.40 | 55.10 | 73.00 | 74.30 | 76.00 | 69.20 | 78.00 | **68.29** |
| T16 | 53.20 | 58.30 | 74.20 | 76.20 | 77.00 | 69.90 | 79.30 | **69.73** |
| T17 | 52.40 | 57.90 | 74.70 | 76.80 | 77.80 | 70.70 | 79.10 | **69.91** |
| AVG | **52.44** | **57.46** | **72.37** | **75.39** | **75.06** | **69.19** | **76.76** |  |

Les meilleurs résultats en fonction des ensembles de données.

|  |  |
| --- | --- |
| T11 | 76.76 |
| T8 | 75.39 |
| T9 | 75.06 |
| T7 | 72.37 |
| T10 | 69.19 |
| T6 | 57.46 |
| T5 | 52.44 |

Les meilleurs résultats en fonction des architectures.

|  |  |
| --- | --- |
| T14 | 70.00 |
| T13 | 69.91 |
| T17 | 69.91 |
| T16 | 69.73 |
| T15 | 68.29 |
| T11 | 65.89 |
| T12 | 65.01 |

Représentation graphique de la meilleure architecture avec le meilleur ensemble de données (twitter14 avec twitter11).

## Analyse des résultats